Daksh Technical / Language Specification

Phase 0 — Frozen Web Interpreter Baseline

Document purpose

This specification records the Phase 0 language model, syntax decisions, runtime behavior, standard-library surface, and implementation boundary established for the Daksh web interpreter. It is intended as a technical reference rather than a beginner tutorial.

Status

Phase 0 is complete and treated as a frozen baseline. Features explicitly marked future, optional, or unfinished are not treated as required Phase 0 syntax.

1. Language Model

Daksh is a programming language using a lexer → parser → AST → interpreter → runtime architecture.

Current execution target: browser-based/web interpreter.

Future target: native compiler/runtime and CLI after the web versions establish usage and feedback.

Source files use the **.dax** extension in the current project.

2. Front-End Architecture

Lexer: converts source text into tokens.

Parser: converts tokens into an abstract syntax tree (AST).

AST: represents statements and expressions structurally.

Interpreter: evaluates the AST and executes statements.

Runtime: stores variables, functions, constants, module values, and runtime behavior.

Web UI: supplies the editor, Run action, and output area.

3. Variables

The primary variable declaration and reassignment form is **set**. Constants use **const**.

Declaration:

```
set name = "Manideep"
```

Reassignment:

```
set name = "Daksh"
```

Multiple declaration:

```
set name = "Manideep", age = 21, score = 95.5
```

Identifiers begin with a letter or underscore and may continue with letters, digits, and underscores. Keywords are reserved. Daksh identifiers are case-sensitive.

4. Scope

An outer variable is visible inside a nested block. A variable created inside a block is not visible after leaving that block.

The Phase 0 runtime therefore supports lexical/block visibility for the tested control-flow scopes.

5. Data Types

Implemented/core value categories are: Integer, Float/Decimal, String/Text, Boolean, Null, List/Array, and Object/Dictionary.

Character and Tuple were identified as optional rather than required Phase 0 types.

Daksh supports dynamic reassignment of a variable to values of different types.

Type inspection: `type(value)`

Conversion examples: `integer("25")` and `text(100)`.

Exact coercion rules are not frozen beyond the currently implemented behavior.

6. Literals and Collections

Lists use square brackets and support indexing and mutation.

```
[10, 20, 30]
```

Objects/dictionaries use key/value pairs.

```
{ name: "Manideep", score: 95 }
```

Nested lists and objects are supported.

Property access uses dot notation; indexed access uses brackets.

Nested assignment is supported.

```
set data.user.name = "Daksh"
```

```
set data.scores[1] = 99
```

```
set matrix[0][1] = 50
```

7. Operators

Arithmetic: +, -, *, /, %, **

Comparison: ==, !=, <, >, <=, >=

Logical: &&, ||, !

Assignment: =

Compound assignment: +=, -=, *=, /=, %=

Increment/decrement: ++, --

Identity and membership operators were identified but their exact syntax was not finalized in Phase 0. Bitwise and ternary operators remain future work.

8. Expressions

Expressions can contain literals, identifiers, arithmetic, comparisons, logical operators, function calls, list indexing, property access, objects, and nested expressions.

Parentheses can be used to control grouping and precedence.

9. Statements

Core statement forms include variable declarations/reassignment, constants, output, conditions, loops, functions, returns, imports, and expression statements.

The language supports standalone increment/decrement expressions such as `i++`.

10. Conditions

Daksh supports **if**, **else if**, and **else**.

```
if score >= 90 { tell "A" } else if score >= 75 { tell "B" } else { tell "C" }
```

11. Loops

while: condition-controlled repetition.

loop: general loop construct supported by the interpreter.

for range: inclusive ascending range syntax.

```
for i in 1..10 { tell i }
```

foreach: iteration over collection values.

```
foreach item in items { tell item }
```

Control: break and continue. Nested loops are supported.

12. Functions

Functions are declared with **function**, can receive parameters, create local values, return values, call other functions, and recurse.

```
function add(a, b) { return a + b } set result = add(10, 20) tell result
```

Functions can return objects and lists. The finalized syntax for default parameters, named parameters, and multiple return values is not part of the frozen Phase 0 contract.

Anonymous functions/lambdas are future work.

13. Input / Output

tell prints a value to the output panel.

```
tell "Hello Daksh"
```

input obtains user input through the web runtime.

```
set name = input("Enter your name: ")
```

14. Constants

A value declared using **const** cannot be modified by normal assignment.

```
const pi = 3.14
```

Attempting to modify it produces a runtime error.

15. Modules & Imports

Modules are loaded using **import**.

Examples include:

```
import math
import string
import list
import random
import time
import system
import file
```

User-defined module/export grammar, aliases, and selective-import syntax were not finalized in Phase 0.

16. Error Model

Phase 0 recognizes and handles core error categories including syntax errors, runtime errors, undefined variables, division by zero, invalid list indexes, constant modification, invalid assignment targets, null nested assignment, and import/module errors.

Custom user-defined error syntax is future/not finalized.

17. Runtime Assignment Semantics

Assignment targets can be simple identifiers, object properties, list/object indexes, and nested combinations of those forms.

Examples:

```
set x = 10
set data.name = "Daksh"
set data.scores[1] = 99
set matrix[0][1] = 50
```

18. Standard Library Contract

The Phase 0 implementation includes the following modules and tested APIs.

18.1 math

abs, sqrt, pow, floor, ceil, round, max, min, pi, e

18.2 string

length, upper, lower, trim, contains, startsWith, endsWith, replace

18.3 list

length, add, remove, contains, first, last

18.4 random

number, integer, choice

18.5 file

read, write, download are browser-safe workflows using file-picker/save/download behavior. exists is intentionally unavailable for arbitrary filesystem inspection in the browser.

18.6 time

today, now, nowDateTime, format, addDays, removeDays, isBefore, isAfter, dayName, monthName, yearNumber, monthNumber, dayNumber

18.7 system

osName, osVersion, deviceName, userName, language, architecture, dakshVersion, memory, batteryLevel, isOnline, sleep, exit, restart

19. Date/Time Formatting

The current time formatting implementation recognizes the documented pattern components such as YYYY, MM, DD, HH, mm, and ss.

20. Browser Runtime Boundary

Some operations are constrained by browser security. File access uses user-mediated browser APIs rather than unrestricted filesystem access. System information is limited to values the browser can expose. A native runtime is planned for a later phase.

21. Memory Model

Phase 0 defines the conceptual model of variables, stack/heap behavior, references, and garbage collection, but the native memory/GC implementation is not frozen. This remains an implementation boundary for the future native toolchain.

22. Compiler / Interpreter Boundary

Phase 0 is complete for the web interpreter architecture. A native compiler, native runtime, CLI, and related tooling are later-phase work.

23. Reserved Keywords

Core reserved words: tell, set, const, input, if, else, loop, while, for, break, continue, function, return, import, true, false, null, foreach.

24. Phase 0 Non-Frozen / Future Items

The following are intentionally outside the frozen Phase 0 contract: exact coercion rules; native memory/GC model; default/named/multiple-return syntax; user-defined module/export syntax; import aliases/selective imports; exact identity/membership operators; custom errors; Character/Tuple as required types; ternary/pattern matching; bitwise operators; anonymous/lambda functions; native compiler/runtime; CLI; VS Code extension; and other future tooling.

25. Reference Execution Flow

```
Daksh source → Lexer → Tokens → Parser → AST → Interpreter → Runtime → Browser output
```

This is the canonical Phase 0 execution pipeline for the current web implementation.

26. Conformance Guidance

A Phase 0 implementation should preserve the documented core syntax and behavior, including variable operations, control flow, functions, collections, modules, standard-library APIs, scope, constants, nested assignment, and core error behavior. Future syntax should not be treated as required for Phase 0 conformance.

End of Phase 0 Technical Specification